BIG DATA: Clústering de documentos a partir de métricas de similitud

Tópicos Especiales en Telemática

Pablo Quijano Jaramillo

pquijano@eafit.edu.co

Sebastian Galeano Cañas

sgalean7@eafit.edu.co

Profesor:

Edwin N. Mtoya Munera

emontoya@eafit.edu.co

Universidad Eafit

Medellín

23 de noviembre de 2017

1. Palabras clave

1.1. Apache Spark:

Apache Spark es un sistema de computación en clúster rápido y de uso general. Proporciona API de alto nivel en Java, Scala, Python y R, y un motor optimizado que admite gráficos de ejecución general. También es compatible con un amplio conjunto de herramientas de alto nivel que incluyen Spark SQL para SQL y procesamiento de datos estructurados, MLlib para aprendizaje automático, GraphX para procesamiento de gráficos y Spark Streaming.

1.2. RDD(Resilient Distributed Dataset):

La abstracción de datos en Spark se realiza por medio de los RDD. Un RDD es una colección de objetos de solo lectura particionada sobre un conjunto de máquinas que puede ser reconstruida si se pierde alguna de las particiones.

1.3. K-means:

Método de agrupamiento, que tiene como objetivo la partición de un conjunto de n observaciones en k grupos en el que cada observación pertenece al grupo cuyo valor medio es más cercano.

1.4. Data Mining:

Campo de la estadística y las ciencias de la computación referido al proceso que intenta descubrir patrones en grandes volúmenes de conjuntos de datos.

1.5. Data Set:

Conjunto de datos.

1.6. Text Mining:

Se refiere al proceso de derivar información nueva a partir de textos.

1.7. TF(Term Frequency):

Es la forma mas sencilla de transformar las cadenas a números. Cada palabra es remplazada por su número de apariciones en el texto. Si t es un termino y d un documento : TF(t,d).

2. Introducción

El proyecto 4 consistió en diseñar e implementar un algoritmo que permitiera agrupar (clustering) un conjunto de documentos apoyándonos en la libreria MLlib para machine learnig de spark, de esta librería se utilizo la implementación de el algoritmo para clustering de k–means y una métrica de similaridad entre documentos. Para encontrar la similitud entre los documentos utilizamos el algoritmo de TF-IDF, que nos permite no solo saber la frecuencia con que una palabra se encuentra en un documento sino también su importancia. El dataset utilizado para las pruebas fue el de Gutenberg, que cuenta con alrededor de 3000 documentos; teniendo estos documentos la idea además de resolver el problema planteado era analizar las diferencias entre una solucion para HPC vs una implementación basada en Big Data.

3. Resumen

El análisis de textos o text mining permite obtener información a partir de textos en este caso a partir de un conjunto de textos, el programa nos permite agrupar diferentes textos para así poder mostrar todos los textos que se parezcan entre si, esto podría permitir hacer recomendaciones a las personas o compañías sobre temas que puedan ser de su interés según lo que van viendo cada día. En este documento se encontrara un análisis detallado de la solución dada, además se compararan los resultados con los de la práctica pasada de HPC, y se darán conclusiones al respecto.

4. Marco Teórico

Se tiene un conjunto de documentos grande D, el cual contiene di documentos de texto, de algún corpus o dataset dado, en este caso se usara la base de textos de Gutenberg que se encuentra en la pagina: http://www.gutenberg.org
Las características que debe contener el dataset es que presente relaciones sintácticas y semántica entre los documentos, por ejemplo en un dataset de noticias presenta una correlación entre documentos de un mismo tipo de noticia (género).
El problema básicamente se sintetiza en 2 subproblemas:

4.1. Subproblema 1:

Diseño e implementación de una función de similaridad entre 2 documentos, esto es: dato di y dj , se define una función de de similaridad, fs, entre di y dj como: fs(di; dj) = [minval; maxval]. Esto lo logramos usando el algoritmo de TF-IDF.

4.2. Subproblema 2:

Una vez se tiene definida e implementada la función de similaridad, se ejecuta el algoritmo de agrupamiento (clustering), que permite dividir el conjunto de documentos

en un número de subgrupos. Estos algoritmos de clustering pertenecen a la categoría de Aprendizaje de Máquina NO supervisado. el algoritmo usado fue k–means, el cual permite dividir el conjunto de documentos en k subgrupos.

5. Análisis y diseño

Básicamente se emplearon 2 algoritmos, el kMeans para agupar los documentos, y el TF-IDF para medir la similaridad entre estos; al tf-idf se le pasa el dataset para que este haga las comparaciones de las palabras entre los documentos y encuentre las palabras con mayor frecuencia y relevancia; despues se usa lo que devuelve el tf-idf en el algoritmo de Kmeans dando como resultado el numero de clusters deseados con los documentos agrupados.

6. Implementación

6.1. Tecnologias:

El lenguaje de programación utilizado fue python en su version 2.7 y en especial usamos la biblioteca de MLlib de Spark que nos permitio convertir el datasets en un tipo de datos RDD para sus posteriores operaciones

6.2. TF-IDF(Term frequency-inverse document frequency):

Es un método de conversión a vector ampliamente utilizado en la mineria de texto para reflejar la importancia de un termino de un documento en el datasets
IDF representa la inversa de las apariciones de un termino en el datasets. Si consideramos un termino t y un documento D, DF(t,D) seria el numero de documentos en los que aparece t

$$IDF(t, D) = \log \frac{|D| + 1}{DF(t, D) + 1}$$

TF-IDF es la multiplicación de los dos conceptos TF e IDF

$$TFIDF(t, d, D) = TF(t, d) \cdot IDF(t, D).$$

```
hashingTF = HashingTF() # Crea un
tf = hashingTF.transform(valores)
idf = IDF(minDocFreq=5).fit(tf)
tfidf = idf.transform(tf) # Calcu
```

6.3. Kmeans:

El algoritmo K-means es uno de los algoritmos de aprendizaje no supervisado más simples para resolver el problema de la clusterización. El procedimiento aproxima por etapas sucesivas un cierto número (prefijado) de clusters haciendo uso de los centroides de los puntos que deben representar.

```
clusters_model = KMeans.train(tfidf, k, maxIterations=iteraciones)
clusters = clusters_model.predict(tfidf).collect()  # Encuentra el
```

7. Análisis de resultados

Se hizo un análisis de los resultados obtenidos en esta práctica, comparándolos con los de la práctica anterior de HPC, que era el mismo problema pero resuelto en computación paralela. Se obtuvieron los siguientes resultados:

7.1. 50 archivos:

HPC: 30 segundos. BigData: 49 segundos.

7.2. 100 archivos:

HPC: 76.5 segundos (1.2 minutos) BigData: 59 segundos (1.9 minutos)

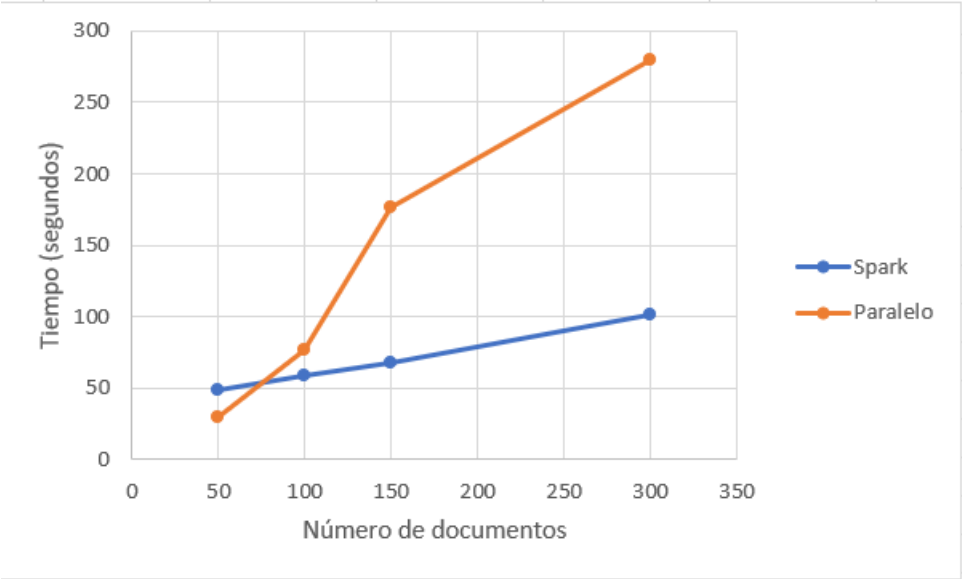
7.3. 150 archivos:

HPC: 176.7 segundos (2.9 minutos) BigData: 68 segundos (4 minutos)

7.4. 300 archivos:

HPC: 280 segundos (4.6 minutos) BigData: 101 segundos (6.7 minutos)

7.5. Gráfica



- ✓ Podemos observar que con la solución de BigData se obtienen tiempos mucho inferiores al ejecutar el programa que los que se obtienen con HPC.

1. Se pudo analizar tanto el problema de HPC vs Big Data obteniendo los resultados de cada uno y tomando sus tiempos.
2. Para obtener resultados mas precisos se requeriría conocer una k con antelación que sea mas precisa para el algoritmo de K-means.
3. El tiempo varia de acuerdo al numero de documentos que se lean del dataset.
4. Se observa como para este caso de clustering con una cantidad de datasets tan grande Utilizar el framework de ejecución y procesamiento distribuida basado en Spark en un Cluster Hadoop/Spark es mas efectivo que procesamiento distribuido HPC

8. Referencias

- ✓ IF-TDF, available at: https://spark.apache.org/docs/2.2.0/ml-guide.html [Accessed 22 Nov. 2017].
- ✓ K-MEANS CLUSTERING, available at: https://spark.apache.org/docs/2.2.0/mllib-clustering.html [Accessed 22 Nov. 2017].